REPORT DOCUMENTATION PAGE

1. REPORT DATE *(DD-MM-YYYY)* 7 Jul 15	2. REPORT TYPE Journal Article	3. DATES COVERED *(From – To)* Dec 2014 – Apr 2015
4. TITLE AND SUBTITLE A Tool that Uses the SAS® PRX Functions to Fix Delimited Text Files		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Paul Genovesi		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) USAF School of Aerospace Medicine PHR 2510 Fifth St. Wright-Patterson AFB, OH 45433-7913		8. PERFORMING ORGANIZATION REPORT NUMBER AFRL-SA-WP-JA-2015-0028
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution A: Approved for public release; distribution is unlimited. Case Number: 88ABW-2015-1635, 31 Mar 2015

13. SUPPLEMENTARY NOTES
Published in SAS Global Forum 2015 Conference Proceedings

14. ABSTRACT

Delimited text files are often plagued by appended and/or truncated records. Writing customized SAS® code to import such a text file and break out into fields can be challenging. If only there was a way to "fix" the file before importing it.

Enter the file_fixing_tool, a SAS® Enterprise Guide® project that uses the SAS PRX functions to import, fix, and export a delimited text file. This fixed file can then be easily imported and broken out into fields.

The file_fixing_tool project was written using SAS Enterprise Guide 5.1 and SAS 9.3 on a Windows 7 Enterprise operating system.

15. SUBJECT TERMS
Delimited files, delimited text files, fix, repair, truncated, truncated records, appended, appended records, regular expressions, sas, perl, prx, prx functions, sas enterprise guide, sas eg, prxmatch, prxchange

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Paul Genovesi
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	SAR	21	19b. TELEPHONE NUMBER *(include area code)*

A Tool that Uses the SAS® PRX Functions to Fix Delimited Text Files

Paul Genovesi, Henry Jackson Foundation for the Advancement of Military Medicine, WPAFB, OH

ABSTRACT

Delimited text files are often plagued by appended and/or truncated records. Writing customized SAS® code to import such a text file and break out into fields can be challenging. If only there was a way to "fix" the file before importing it.

Enter the file_fixing_tool, a SAS® Enterprise Guide® project that uses the SAS PRX functions to import, fix, and export a delimited text file. This fixed file can then be easily imported and broken out into fields.

The file_fixing_tool project was written using SAS Enterprise Guide 5.1 and SAS 9.3 on a Windows 7 Enterprise operating system.

INTRODUCTION

Delimited text files can be considered "broken" when they contain appended and/or truncated records.

It is common knowledge that SAS's PRX functions are powerful tools for turning messy text into structured data. But can they be used in a program for restructuring a broken delimited text file into a file that can be imported into SAS? This paper will show the answer to this question is yes.

The following topics will be discussed:

- Defining a delimited text file

- Delimited text file structure (both normal and broken)

- Text qualifying

- The truncated-only method for fixing broken delimited text files containing no appended records

- The appended method for fixing broken delimited text files containing appended and/or truncated records

- Common surrounding characters (CSCs)

- Setting up the file_fixing_tool

- Running the file_fixing_tool

- Testing the file_fixing_tool using one of the 33 test case files

- Before and after examples of fixed delimited text test files

- Helpful reminders and other important information

DEFINING A DELIMITED TEXT FILE

Although delimited text files exist in a wide variety of formats, they all contain the same basic structure of records separated by record separators and the record's fields separated by field delimiters. Common record separators are a newline character, a carriage return character, or a combined newline and carriage return character. Common field delimiters are comma, tab, colon, semicolon, pipe, caret, etc.

A very common delimited text file format is the CSV (comma-separated values) format, but it has various specifications and implementations (Shafranovich, 2005). Shafranovich's RFC4180 document (http://tools.ietf.org/html/rfc4180) contains a definition for the CSV format that, as he states, "seems to be followed by most implementations." His definition consists of the following rules:

Note: For the below rules, CRLF = the record separator and comma = the field delimiter

1. Each record is located on a separate line, delimited by a line break (CRLF). For example:
   aaa,bbb,ccc CRLF
   zzz,yyy,xxx CRLF
2. The last record in the file may or may not have an ending line break. For example:
   aaa,bbb,ccc CRLF
   zzz,yyy,xxx
3. There may be an optional header line appearing as the first line of the file with the same format as normal record lines. This header will contain names corresponding to the fields in the file and should contain the same number of fields as the records in the rest of the file (the presence or absence of the header line should be indicated via the optional "header" parameter of this MIME type). For example:
   field_name,field_name,field_name CRLF
   aaa,bbb,ccc CRLF
   zzz,yyy,xxx CRLF
4. Within the header and each record, there may be one or more fields, separated by commas. Each line should contain the same number of fields throughout the file. Spaces are considered part of a field and should not be ignored. The last field in the record must not be followed by a comma. For example:
   aaa,bbb,ccc
5. Each field may or may not be enclosed in double quotes (however, some programs, such as Microsoft Excel, do not use double quotes at all). If fields are not enclosed with double quotes, then double quotes may not appear inside the fields. For example:
   "aaa","bbb","ccc" CRLF
   zzz,yyy,xxx
6. Fields containing line breaks (CRLF), double quotes, and commas should be enclosed in double quotes. For example:
   "aaa","b CRLF
   bb","ccc" CRLF
   zzz,yyy,xxx
7. If double quotes are used to enclose fields, then a double quote appearing inside a field must be escaped by preceding it with another double quote. For example:
   "aaa","b""bb","ccc"

For the purposes of the file_fixing_tool, we need to add two additional rules:

1. Other characters can be used as the field delimiter. The following field delimiters have been tested using the file_fixing_tool:
   a. comma
   b. caret
   c. colon
   d. semicolon
   e. pipe
   f. tab

2

Note: If your broken delimited text file uses a field delimiter other than one of the above six, then please see the first bullet in the section labelled "Helpful reminders and other important information."

2. Either the double quote or the single quote can be used for text qualifying (i.e., enclosing) a field and they both cannot be used for text qualifying within the same file.

Your broken delimited text file must adhere to the above 9 rules. Otherwise, there is no guarantee that the file_fixing_tool will fix your broken file.

DELIMITED TEXT FILE STRUCTURE (BOTH NORMAL AND BROKEN)

As previously stated, a delimited text file contains records that are separated by record separators, and these records contain fields that are separated by field delimiters. An appended record occurs when a record is not immediately followed by a record separator. A truncated record occurs when a record contains a record separator within the record. Sometimes a delimited text file contains both appended and truncated records. Let's use the following examples to illustrate.

If we were to visualize an unbroken delimited text file containing no appended records and no truncated records, it would resemble the following where [-----] is one complete record and <rs> is the record separator:

[-----]<rs>

[-----]<rs>

[-----]<rs>

[-----]<rs>

Notice that the file's record separator is correctly located at the end of each record, which causes each line to contain exactly one record.

A broken delimited text file containing only appended records would resemble this:

[-----][-----]<rs>

[-----][-----][-----]<rs>

[-----]<rs>

[-----][-----]<rs>

Notice that the lack of record separators after records 1, 3, 4, and 7 causes the appended records.

A broken delimited text file containing only truncated records would resemble this:

[----<rs>

-]<rs>

[-----]<rs>

[---<rs>

--]<rs>

[-----]<rs>

Notice that record separators occurring within records 1 and 3 cause the truncated records.

And lastly, a broken delimited text file containing both appended and truncated records would resemble this:

[-----][----<rs>

-]<rs>

3

[-----]<rs>

[---<rs>

--][-----][-----]<rs>

[---<rs>

--]<rs>

Notice that the lack of record separators after records 1, 4, and 5 causes the appended records while record separators occurring within records 2, 4, and 7 cause the truncated records.

TEXT QUALIFYING

Only the double quote or single quote can be used as a text qualifier to surround a cell (i.e., an individual field contained within one record), and both cannot be used as text qualifiers within the same delimited text file.

Text qualifying MUST take place if either of the following two conditions occurs:

1. A delimited text file contains a cell in which there exists a field delimiter. A cell with this condition must be text qualified (i.e., contain surrounding text qualifiers).

2. A delimited text file contains a text-qualified cell in which there exists the text-qualifier character. A text-qualifier character occurring in such a cell must be escaped with another text-qualifier character. In other words, it takes two text-qualifier characters to mean one (i.e., two double quotes to mean one double quote and two single quotes to mean one single quote). A single unescaped text-qualifier character occurring within a text-qualified cell will throw off the column alignment for that record when the delimited text file is imported into SAS.

Note: A cell CAN be text qualified without meeting one of the above two conditions.

TRUNCATED-ONLY METHOD

This method can only be used on broken delimited text files containing truncated records and no appended records. But what is the definition of an appended record?

Within a broken delimited text file, an appended record occurs when the beginning of any record occurs on a line containing any characters from the previous record. In other words, a record separator does not exist between two records (as it should).

Besides the "no appended records" requirement, there is one other requirement for using this method, and that requirement is the following:

Within the broken text file, the first field of every record and its immediately following field delimiter (i.e., the field delimiter occurring between the first and second fields) must occur on the same line.

In other words, a record separator cannot exist between a record's first field and its immediately following field delimiter. Without this requirement, the tool will not be able to determine when it has encountered a new record and will treat the new record's first field fragment as part of the previous record's last field. The chances of the first field being so long that it would be truncated are slim, since the vast majority of delimited text files locate their identifier or key variables among the beginning fields while their variables containing long text strings are located among the ending fields.

Unlike the appended method, the truncated-only method does not use your last_field and first_field patterns (i.e., the patterns contained in your two macro variables) to fix your delimited text file. It uses a built-in regular expression to identify and count the number of fields encountered up until the last field, at

4

which point another regular expression is used to deal with separating the record's last field from the following record's first field.

APPENDED METHOD

Fixing a broken delimited text file containing only truncated records is relatively easy. Fixing one containing appended records (with or without truncated records) is more difficult and sometimes impossible.

The key to fixing a delimited text file containing appended records is developing a pattern (i.e., a perl regular expression segment) that isolates either the file's first field or a pattern that isolates its last field from the rest of the file. (The file's other fields are already isolated by their surrounding field delimiters.) In other words, with this pattern, we are able to identify throughout the file the location at which one record ends and the next record begins. Identifying this location is critical because this is the only location throughout the text file that should contain a record separator instead of a field delimiter, which would be expected since, as we have mentioned, this location is the end of one record and the beginning of another. In fact, any delimited text file with appended records can be fixed if this location can be identified no matter how appended or truncated the records in your file may be.

CREATING YOUR LAST_FIELD AND FIRST_FIELD PATTERNS FOR THE APPENDED METHOD

To identify and isolate your broken file's last field from its first field, you need to enter patterns in the form of perl regular expression segments for the macro variables last_field and first_field. These patterns will depend on your knowledge of the field's contents and how close your pattern comes to matching those contents.

Here are a few examples of field contents and their matching patterns.

NOTE: The term "visibly blank" means having 0 or more spaces.

1. Social Security number
    a. Contents: a 9-digit character string
    b. Pattern: \d{9}
2. Social Security number with some cells being visibly blank
    a. Contents: a 9-digit character string or visibly blank
    b. Pattern: (\d{9}| *) where pipe is the "or" metacharacter and <space>* means 0 or more spaces
    c. Note: The file_fixing_tool requires the use of "non-capturing group" parentheses whenever parentheses are used within your last_field and first_field patterns, so the pattern would have to be entered as:

        (?:\d{9}| *)
3. Categorical data
    a. Contents: A string containing either red, white, blue, or visibly blank
    b. Pattern: (?:red|white|blue| *)

        Note#2: Non-capturing group parentheses MUST always surround an "or" operation or a continuous series of "or" operations like the above pattern.
4. Categorical data (case-insensitive)
    a. Contents: A string containing red, white, blue, RED, WhiTE, BluE, etc…, or visibly blank
    b. Pattern: (?i:red|white|blue| *)

5

Note: The 'i' in '(?i:' makes the contents case-insensitive (Dunn, 2011).

5. A number between 1 and 1000000 (one million)

   a. Contents: A character string containing between 1 and 7 digits or visibly blank

   b. Pattern: (?:\d{1,7}| *)

6. Date (using a very forgiving pattern)

   a. Contents: A date string with format MM/DD/YYYY or visibly blank

   b. Pattern: (?:\d{2}\/\d{2}\/\d{4}| *)

   Note: Use more "forgiving" patterns when your data are messy and error laden. For example, the above pattern will match the string 55/77/9999, but that is ok, since our goal is to fix the broken file and not identify bogus date data. If our pattern is too precise, like the next example, then the file_fixing_tool will fail.

   Note#2: Although the forward slash is not a metacharacter, it must be escaped (i.e., back-slashed), since the file_fixing_tool uses it as a regular expression delimiter in its processing.

7. Date (incorrect use of a precise and not very forgiving pattern)

   a. Contents: A date string with format MM/DD/YYYY or no contents

   b. Pattern: (?:(?:0?[1-9]|1[012])\/(?:0?[1-9]|[12][0-9]|3[012])\/(?:19|20)\d\d| *)

   Note: WRONG! Precision like this will cause the file fixing tool to fail, since the pattern will match a real-life date and will not match a string like 55/77/9999. The file_fixing_tool needs to be able to match any existing junk data if it exists with the correct format. Save this kind of precision for downstream programming involving identifying bogus dates. It has no place in the fixing of broken delimited text files.

COMMON SURROUNDING CHARACTERS (CSCs)

Common surrounding characters (CSCs) are characters that can exist on the edges of your field (i.e., cell) yet still are contained within the field's contents.

For text-qualified fields:

1. They can exist just inside the text qualifiers surrounding your field.
2. They can be any of the following characters:
   a. The field delimiter character
   b. A pair of text-qualifier characters (meaning a successive EVEN number of text-qualifier characters) that does not include the field's two actual surrounding text qualifiers
   c. The space character
   d. The text qualifier character NOT being used as the text qualifier. For example, if the double quote was being used as the text qualifier, then the one not being used would be the single quote and vice versa.

For non text-qualified fields:

1. They can exist just inside the field delimiters.
2. They can be any of the following characters:
   a. The double quote character
   b. The single quote character
   c. The space character

CSCs can occur in any of your broken file's fields and they do not need to be accounted for within your last_field or first_field patterns when running the appended method.

So what are the benefits of automatically accounting for CSCs in your last field's or first field's contents? There are several benefits:

1. These characters tend to occur quite often on the edge of fields whether the field is considered to be text qualified or not.
2. If the file_fixing_tool did not automatically account for them occurring in your last field or first field contents, then they would need to be accounted for in your last_field and first_field patterns. By accounting for them, you are always covered.
3. Accounting for them does not affect the match since the pattern will account for them only when they are present in your field.

   Note: It is extremely important to remember that only CSCs occurring on the edges of the field contents (i.e., outside any non-CSC characters) are automatically accounted for by the file_fixing_tool. If they occur inside of non-CSC characters, then they need to be accounted for within the field patterns contained in your last_field or first_field macro variables.

SETTING UP THE FILE_FIXING_TOOL

The following are steps for setting up the file_fixing_tool:

1. Create a directory named "file_fixing_tool_dir" on your desktop.

2. Within this directory, create the following two subdirectories:

   a. infile_dir – will contain broken delimited text files of the following types:

      1) Test case files that come with the file_fixing_tool (33 test case files)

      2) Your own broken delimited text files that need fixing

   b. results_dir – fixed delimited text files will be sent here and be named one of the following:

      1) APPEND_METHOD_RESULTS.txt

      2) TRUNCATED_ONLY_METHOD_RESULTS.txt

3. Copy the file_fixing_tool SAS Enterprise Guide project to the file_fixing_tool_dir directory.

4. Copy all 33 test case files to the infile_dir directory.

5. Open the file_fixing_tool SAS EG project.

6. Within the following three process flows, change the path of the imported fixed file to the FULL PATH of your results_dir directory:

   a. Appended Method Testing, NoText Qualifying
   b. Appended Method Testing, Text Qualifying
   c. Truncated-Only Method Testing

7. Open the process flow named "Set macro variables and compile macros" and open the program named "everything_macro" and enter your path for the macro variable named dir. Make sure to include the backslash at the end of your path. For example:

   ```
   C:\Users\genovePC\Desktop\file_fixing_tool\
   ```

8. Setup is now complete.

RUNNING THE FILE_FIXING_TOOL

The header record (if one exists at the top of your broken text file file) must be removed and stored before running the file_fixing_tool, since it will interfere with the regular expression matching. After you have run

7

the file_fixing_tool and fixed your broken file, simply prepend the header record back to the top of the fixed file. I, the author of this paper, apologize for not adding this functionality. I simply ran out of time. Coding this functionality is a lot trickier than it would seem and the costs might outweigh the benefits.

To run the file_fixing_tool, do the following:

1. Open SAS Enterprise Guide and open the file_fixing_tool.

2. Open the process flow named "Set macro variables and compile macros" and open the program named "everything_macro."

3. You are only allowed to modify the following macro variables contained in the area within the "MODIFIABLE CODE" boundaries:

```
/*##########################
 # MODIFIABLE CODE:  BEGIN  #
 ##########################*/
 %let contains_text_qualifying = %str(y);
 %let fld_dlm       = %nrstr(\x2C);       /* Identify the field delimeter.  Use \x2C for comma */
 %let num_fields    = %str(5);          /* Number of fields in the file.  */
 %let txq           = %str(%");      /* %" for double quote, %' for single quote, or n for no text qualifying */
 %let run_appended_method        = %str(n); /* y = run, n = don't run */
 %let run_truncated_only_method = %str(y);  /* y = run, n = don't run */
 %let input_line_stop  = %nrstr(=~##);   /* Used for marking the ends of inputted text file lines */
 %let in_file_lrecl    = %str(500);      /* Length of character variables used in processing */
 %let out_file_lrecl   = %str(1000);     /* Length of output file records */
 %let pre_trunc_num_spaces = %str(20);      /* Used for truncation detection */
 %let dir          = %nrstr(C:\Users\genovePC\Desktop\file_fixing_tool\);
 %let infile_dir   = %str(&dir)%str(infile_dir\);
 %let results_dir = %str(&dir)%str(results_dir\);
 %let in_file      = %nrstr(TCases_SomeTxq_Empty_Spaces_EmbDlm_EmbTxq_trunc.txt);  /* */

 %let first_field = %nrstr((?:(?i:[A-Z]+?)| *?));    /* used only for the appended method */
 %let last_field  = %nrstr((?i:red|white|blue| *)); /* used only for the appended method */
/*##########################
 # MODIFIABLE CODE:  END  #
 ##########################*/
```

Notice that the last two macro variables (first_field and last_field) are used by the appended method only and are ignored by the truncated-only method.

4. Set each macro variable according to the following definitions:

   a. contains_text_qualifying − set to y or n.

      1) If set to y, then the file_fixing_tool will process the file as a text-qualified file. You must also set the macro variable txq to the text qualifier being used.

      2) If set to n, then the file_fixing_tool will process the file as a delimited, non-text-qualified file, and the txq macro variable will not be used during processing.

   b. fld_dlm − field delimiter used in your broken file. For example: , <tab> : ; | ^ etc…

      Note: When your field delimiter is a comma, use its hex value (i.e., \x2C). Otherwise, it will be misinterpreted within the %nrstr() macro function as a separator of arguments to the macro function.

   c. num_fields − number of fields contained in your broken file.

   d. txq − the text qualifier used (if file contains text qualifying). Use one of the following :

1) %" – double quotes used as text qualifier

2) %' – single quotes used as text qualifier

3) n – no text qualifying

e. run_appended_method – you must enter either y or n.

f. run_truncated_only_method – you must enter either y or n.

   Note: You cannot run the truncated-only method on a broken, delimited text file containing appended records. Doing so will create bogus results.

g. input_line_stop – keep the already-entered string of =~## This string is used during the importing of your broken file to mark the end of each line (i.e., where a record separator (aka line stop) occurred). If your file happens to contain the string =~##, then choose another string, but be careful since you need to choose characters that are not regular expression metacharacters or characters that are not special in the SAS Macro programming language such as % &.

h. in_file_lrecl – a number used to make sure that truncation does not occur during the importing of your broken file. If you receive a truncation error, then increase this value.

i. out_file_lrecl – a number used to make sure that the fixed record outputted to your fixed file is not truncated. It is also used as length values for certain variables used in creating your fixed record.

j. pre_trunc_num_spaces – a number used for detecting truncation.

k. dir – the directory path containing the directories infile_dir and results_dir.

l. infile_dir – the directory containing your broken, delimited text file.

m. results_dir – the directory where your fixed, delimited text file will be sent.

n. in_file – the name of your broken, delimited text file. Don't forget the ".txt" suffix in your file name.

o. first_field – a pattern (in the form of a perl regular expression segment) that matches the contents of the first field but not the contents of the previous record's last field.

p. last_field – a pattern (in the form of a perl regular expression segment) that matches the contents of the last field but not the contents of the next record's first field.

5. You are now ready to run the file_fixing_tool by clicking the following:

   File → Run file_fixing_tool

TESTING THE FILE_FIXING_TOOL USING ONE OF THE 33 TEST CASE FILES

There are 33 test cases in the form of delimited text files that are used for testing the file_fixing_tool.

Each test case file:

- contains 5 fields

- contains 21 records

- contains a list within its contents (see below picture) of macro variable settings needed for running the file_fixing_tool on the test case file

10

```
TCase_003_comma_FF_uc_LF_lc_EmptyFlds_Spaces - Notepad

File  Edit  Format  View  Help

DECEMBER,---aaa,     First Field contents:  all uc  empties  spaces    ,ccc, blue
   JANUARY,--aaa,    Last Field contents:   all lc  empties  spaces    ,ccc,red
,-----------aaa,     %let contains_text_qualifying = %str(n);          ,ccc,
MARCH  ,----aaa,     %let fld_dlm            = %nrstr(\x2C);           ,ccc,  blue
   ,-------aaa,      %let num_fields         = %str(5);                ,ccc,
MAY,--------aaa,     %let txq                = %str(n);                ,ccc,white
,-----------aaa,     %let run_appended_method      = %str(y);         ,ccc,
,-----------aaa,     %let run_truncated_only_method = %str(y);         ,ccc,  red
   ,-------aaa,                                                        ,ccc,
SEPTEMBER,--aaa,     %let first_field  = %nrstr((?:[A-Z]+| *));       ,ccc,blue
OCTOBER  ,--aaa,     %let last_field   = %nrstr((?:[a-z]+| *));       ,ccc,red
NOVEMBER,---aaa,                                                      ,ccc,white
DECEMBER,---aaa,     %let in_file = %nrstr(                           ,ccc,
   ,-------aaa,      TCase_003_comma_FF_uc_LF_lc_EmptyFlds_Spaces.txt);  ,ccc,red
FEBRUARY,---aaa,                                                      ,ccc,  white
   MARCH,----aaa,                                                     ,ccc,blue
APRIL   ,---aaa,-------------------------------------------------------,ccc,
MAY,--------aaa,-------------------------------------------------------,ccc,white
JUNE ,------aaa,-------------------------------------------------------,ccc,
   ,-------aaa,-------------------------------------------------------,ccc,red
AUGUST,-----aaa,-------------------------------------------------------,ccc,white

                                                                    Ln 1, Col 1
```

- exists in an "already-fixed" state meaning it contains no appended or truncated records

  Note: The 4 test case files whose names begin with 'TC_T' do not exist in an "already-fixed" state. We will explain why shortly.

Most of our test case files exist in an "already-fixed" state because, as previously stated, the key to fixing a broken delimited text file containing appended records (with or without truncated records) is dependent upon developing last field and first field patterns that identify and isolate the last and first fields. The amount of appending and/or truncating occurring in your broken delimited text file is not a factor when using the file_fixing_tool to fix it.

If your last_field and first_field patterns are successful in identifying and isolating either of these fields, then the "already-fixed" file will remain in this state.

If not, then the file_fixing_tool will destroy this state.

TESTING THE FILE_FIXING_TOOL USING 'TC_T' FILES

The 'TC_T' files are "truncated-only" or "truncated and appended" versions of the files whose test case number they contain in their name. For example, the following two files are identical except that one of them is in an "already-fixed" state and the other contains truncated records:

TCase_008_comma_FF_uc_lc_LF_uc_lc_EmptyFlds_Spaces.txt
TC_T__008_TRUNCATED_ONLY_comma_FF_uc_lc_LF_uc_lc_EmptyFlds_Spaces.txt

The 'TC_T' files are used for ensuring that the file_fixing_tool fixes broken delimited text files that actually contain appended records and/or truncated records.

When running the file_fixing_tool on one of the 'TC_T' files, set the macro variables to the same values as the number of the test file contained in the 'TC_T' test file's name. For example, the following two files should have the same macro variable settings since, as we have already mentioned, they are identical files but one has truncated records:

TCase_008_comma_FF_uc_lc_LF_uc_lc_EmptyFlds_Spaces.txt
TC_T__008_TRUNCATED_ONLY_comma_FF_uc_lc_LF_uc_lc_EmptyFlds_Spaces.txt

You will get the exact same results when running the file_fixing_tool on either of these files since they are the same exact file when they are both in their fixed states.

11

RUNNING THE FILE_FIXING_TOOL ON ONE OF THE 33 TEST CASE FILES

1. Open the test case file and notice how the third field contains information dealing with macro variable settings.

2. Open the everything_macro program and set your macro variables to the settings contained in your test case file.

3. Within the everything_macro program and directly below the "MODIFIABLE CODE" area, you will see another area between labels "TESTING AREA: BEGIN" and "TESTING AREA: END."

4. Within this area, uncomment one %let statement for the first_field macro variable and one for the last_field macro variable for the test case that you are running. Each %let statement has an identifying comment to its right.

5. You are now ready to run the file_fixing_tool on your test case file.

IMPORTING THE FIXED TEST FILE

Whenever the file_fixing_tool is run using the appended method on one of the 33 test case files, the resulting fixed file (contained in APPEND_METHOD_RESULTS.txt) is automatically imported into SAS and broken out into its 5 fields.

Each of the 33 test case files has its own file import process within one of the following two process flows:

1. Appended Method Testing, No Text Qualifying
2. Appended Method Testing, Text Qualifying

The following is a picture of the Project Tree showing the two process flows and their contents.
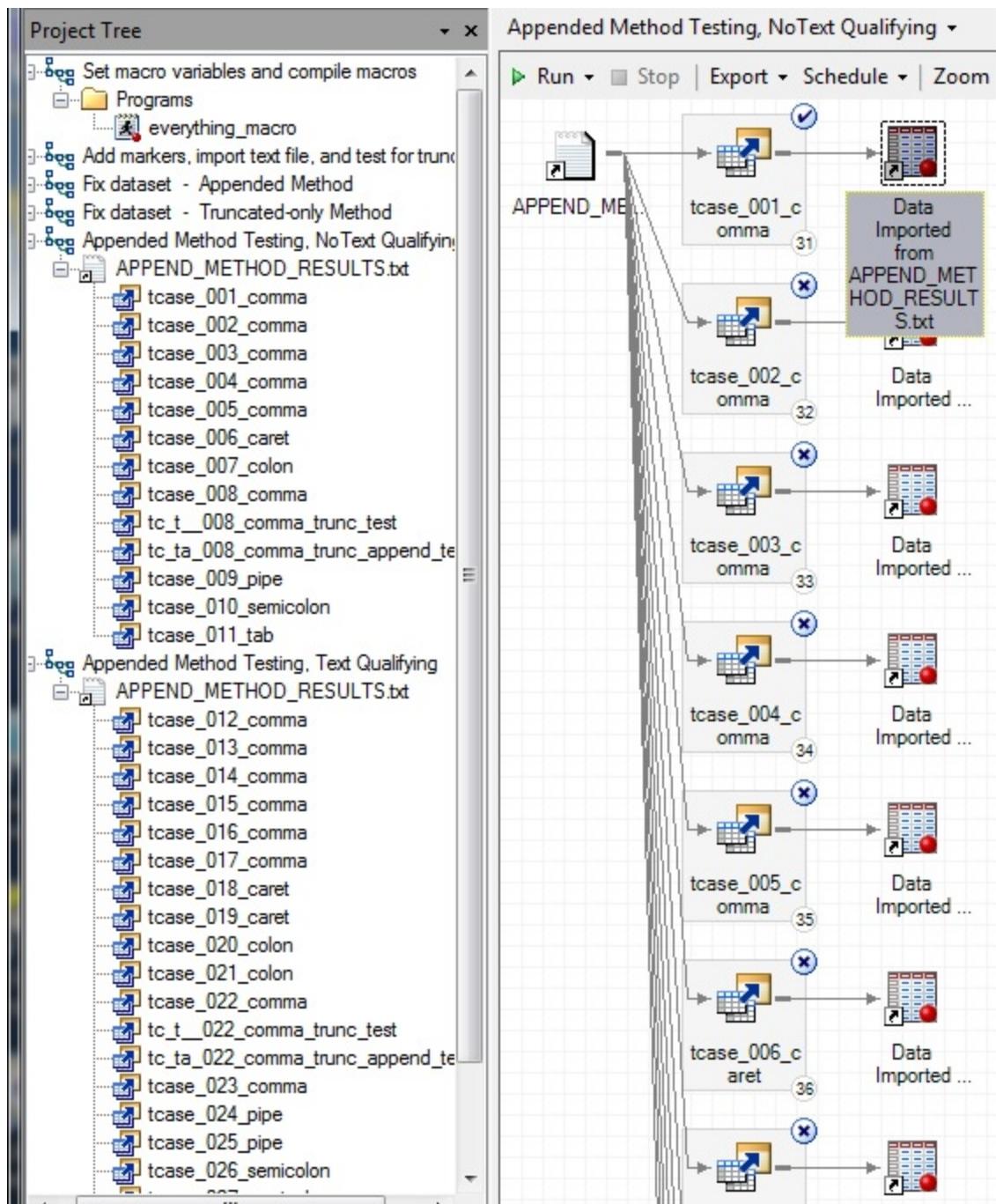
The use of SAS Enterprise Guide's conditional processing ensures that only the import process for the test case file being run will execute.

No test-related import process in either of the above two process flows is executed when the file_fixing_tool is run on a delimited text file other than one of the 33 test files. In other words, every import process within these two process flows is for testing only.

The process flow named "Truncated-Only Method Testing" contains only the following two import processes:
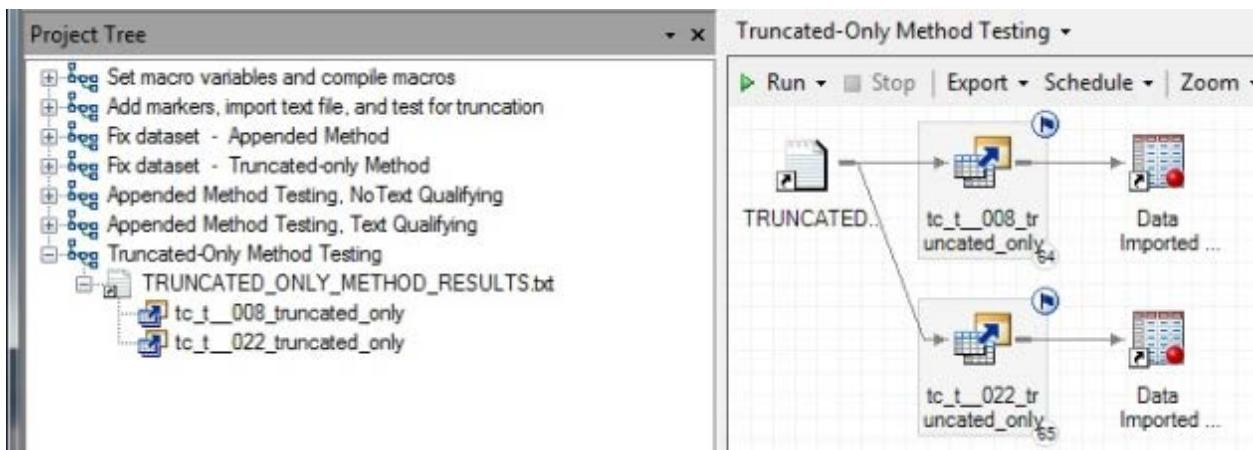
BEFORE AND AFTER EXAMPLES OF FIXED DELIMITED TEXT TEST FILES

EXAMPLE #1

The following broken delimited text file contains truncated records but no appended records:

14

The following is the above broken delimited text file having been fixed using the appended method:

The following is the SAS dataset created by importing the above fixed delimited text file into SAS:

The following is the original broken delimited text file having been fixed using the truncated-only method:

15

The following is the SAS dataset created by importing the above fixed delimited text file into SAS:

EXAMPLE #2

The following broken delimited text file contains both truncated records and appended records:

The following is the above broken delimited text file having been fixed using the appended method. Notice the few double quotes that get matched to one record's last field instead of the following record's first field. This is unavoidable since these double quotes could logically belong to either field.

The following is the SAS dataset created by importing the above fixed delimited text file into SAS:

EXAMPLE #3

The following delimited text file is in an "already-fixed" state since it contains no truncated or appended records:

Here are the results using the appended method that show that the "already-fixed" state has been maintained except for a few double quotes that get matched to one record's last field instead of the following record's first field. This is unavoidable since these double quotes could logically belong to either field.

The following is the SAS dataset created by importing the above fixed delimited text file into SAS:

HELPFUL REMINDERS AND OTHER IMPORTANT INFORMATION

- The file_fixing_tool will not work on space-delimited text files and has only been tested on files containing the six already mentioned delimiters (i.e., comma, caret, colon, semicolon, pipe, and tab). A workaround for space-delimited files would be to use a command or short script to change every occurrence of the space character to one of the above delimiters not already existing in your broken file. If the delimiter already exists, then your workaround would need to employ a bit of indirection in terms of first changing every occurrence of your chosen delimiter to a character not existing (i.e., a non-existing character) in your file and then changing every occurrence of a space character to your chosen delimiter. Then run the file_fixing_tool and fix your file. Now take the fixed text file and change every occurrence of your chosen delimiter back to a space character and change every occurrence of your non-existing character back to your chosen delimiter. You now have a fixed space-delimited text file.

18

- While still on the topic of space-delimited text files, it is worth noting that delimited text files that can be imported using List Input (Base SAS) or Modified List Input (Base SAS), where two consecutive spaces can mean the end of a field, are not the same as space-delimited text files and need to first be turned into space-delimited text files (and then use the steps in the above bullet).

- If you encounter the "missing semicolon" error, then restart the file_fixing_tool.

- Always use the hex value (i.e., \x2C) for the comma character within the following macro variable values in the everything_macro program:

    1. fld_dlm
    2. first_field
    3. last_field

    Failure to do so will result in a "More positional parameters found than defined" error.

- Depending on the length of your first_field and last_field patterns within these macro variables, you might encounter the following warning:

    "WARNING: Your string has more than 262 characters. You might have unbalanced quotation marks."

    Please ignore this warning if your patterns are longer than 262 characters.

- Always remove the header record from the top of your broken delimited text file before running the file_fixing_tool. After running the file_fixing_tool, reattach the header record to the top of your fixed delimited text file.

- Any use of parentheses within your first_field and last_field patterns must be non-grouping parentheses such as (?:…). Otherwise, the capture buffer numbering part of the file_fixing_tool's processing will be destroyed along with your results.

- Any use of the "or" operator within your first_field or last_field patterns must be enclosed in non-grouping parentheses. For example:

    (?:A|B|C)

- Sometimes broken delimited text files contain categorical data values in the last field where these data values begin with characters that other shortened data values begin with. For example, the field might contain the following values: C, CM, CAT, D, DA, DOG.

    The pattern containing multiple "or" operators would need to contain the following order:

    ```
    (?:CM|CAT|C|DOG|DA|D)
    ```

    ```
    If the 'C' in the above pattern occurs before the 'CM' or 'CAT',
    then only the 'C' in 'CAT' will get matched for the last field and
    the 'AT' could get matched to the following record's first field
    depending on your first_field pattern.
    ```

CONCLUSIONS

For the file_fixing_tool to work correctly, your broken, delimited text file must adhere to the 9 rules contained in this paper's section "DEFINING A DELIMITED TEXT FILE."

Knowing which method (appended or truncated-only) to use is important.

If you are able to develop last field and first field patterns that identify and isolate these fields, then use the appended method. If not, then you can still use the truncated-only method as long as your broken file contains only truncated records.

REFERENCES

Dunn T. Grouping, atomic groups, and conditions: creating if-then statements in Perl RegEx. In: Programming beyond the basics. Proceedings of the SAS Global Forum 2011 Conference; 2011 Apr 4-7; Las Vegas, NV. Cary (NC): SAS Institute, Inc.; 2011. Paper 245-2011. [Accessed 1 Mar. 2015]. Available from http://support.sas.com/resources/ papers/proceedings11/245-2011.pdf.

Shafranovich Y. RFC 4180: common format and MIME type for comma-separated values (CSV) Files. 2005. [Accessed 1 Mar. 2015]. Available from http://tools.ietf.org/html/ rfc4180.

ACKNOWLEDGMENTS

RECOMMENDED READING

CSVReader.com. CSV file format. [Accessed 1 Dec. 2014]. Available from http://www.csvreader.com/ csv_format.php.

Genovesi P. Learning SAS's perl regular expression matching the easy way: by doing. [Accessed 1 Mar. 2015]. Available from http://www.sascommunity.org/mwiki/images/1/1b/ RF-10-2014-Learning_SAS%27s_Perl_Regular_Expression_ Matching_the_Easy_Way_By_Doing.pdf.

Wikipedia. Comma-separated values. [Accessed 1 Dec. 2014]. Available from http://en.wikipedia.org/ wiki/ Comma-separated_values.

Windham M. Introduction to regular expressions in SAS®. Cary (NC): SAS Institute, Inc.; 2014.